Trabajo Práctico №2

Métodos de Descenso

Mayo 2017 Optimización

Integrante	LU	Correo electrónico
Hurovich, Gustavo Martín	426/13	gushurovich@gmail.com
López, Agustina Florencia	120/13	agustina.lopez@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria – Pabellón I, Planta Baja
Intendente Güiraldes 2160 – C1428EGA
Ciudad Autónoma de Buenos Aires, Rep. Argentina
Tel/Fax: +54 11 4576 3359
http://exactas.uba.ar

Introducción

Un problema muy estudiado es el de encontrar mínimos de una función. Diversos problemas de optimización consisten o se fundamentan en poder encontrar el mínimo (o máximo) de cierta función objetivo.

Consideraremos en particular una familia de métodos numéricos que permiten resolver este problema, llamada Métodos de Descenso. La idea general consiste en, dado $x_k \in \mathbb{R}^n$, conseguir una dirección de descenso d_k y encontrar un α tal que $x_{k+1} = x_k + \alpha d_k$ cumpla $f(x_{k+1}) < f(x_k)$. Más aún, uno podría buscar el α tal que el descenso sea el mínimo en la dirección d_k elegida. Sería razonable esperar que la sucesión (x_k) converja a un minimizador local de f.

1. Ejercicio 1: Método del Gradiente

1.1. Desarrollo

Nos centraremos ahora en el "Método del gradiente". En él, en cada paso se elige la dirección d_k como la de mayor descenso posible, es decir: $d_k = -\nabla f(x_k)$. Implementaremos este método con cuatro variantes posibles. Recordemos que, en nuestro algoritmo vamos a tener que minimizar $\varphi(\alpha) = f(x_k + \alpha d_k)$. Consideraremos las siguientes tres opciones para hacer esto:

- Búsqueda lineal exacta de paso óptimo
- Búsqueda lineal exacta de paso limitado
- Búsqueda lineal de orden 0

Para el primer caso, utilizamos el comando de MATLAB "*fminsearch*", indicándole únicamente un punto inicial para comenzar a iterar (en nuestro caso, elegimos tal número como el 0).

Para la búsqueda lineal exacta de paso limitado, sólo cambiamos el comando anterior por "*fminbnd*", teniendo que dar ahora un intervalo acotado donde buscar el mínimo. Decidir el tamaño del intervalo no fue sencillo. Si es muy grande, la búsqueda es más costosa, y puede coincidir con la búsqueda de paso óptimo (la cual vimos tiene algunos problemas como el *efecto serrucho*). Por otro lado, si es muy pequeño, necesitamos una gran cantidad de iteraciones para llegar al mínimo, pues en cada una nos acercamos solamente un poco. Luego de probar con varios valores, decidimos escoger uno arbitrario, que funcionaba bien y fijarlo, aunque suponemos que es posible modificarlo en función de características de la función objetivo.

En el último caso, elegimos el método de trisección. Éste parte de un punto inicial x, una dirección d y un intervalo [a,b], que es dividido en tres partes iguales, obteniendo dos nuevos puntos: $x_1 = a + \frac{b-a}{3}$ y $x_2 = a + \frac{2(b-a)}{3}$. Luego, se calcula $\varphi(x_1)$ y $\varphi(x_2)$ para finalmente compararlos y modificar el extremo correspondiente de nuestro intervalo original, volviendo a comenzar el algoritmo con estos nuevos parámetros de entrada. Más específicamente: si $\varphi(x_1)$ es menor a $\varphi(x_2)$, entonces el extremo superior toma el valor de x_2; si es mayor, el inferior pasa a ser x_1; y en el caso de que los valores sean iguales, se considera el intervalo $[x_1, x_2]$.

En criollo, en cada paso deduce en qué tercio(s) del intervalo no está el mínimo, y se queda con lo restante. En cada paso reduce el tamaño del intervalo a analizar por un factor de al menos $\frac{2}{3}$. Finalmente, el método converge al mínimo que buscamos. Como el criterio de parada de nuestro algoritmo es que la distancia entre los extremos del intervalo sea suficientemente chica, el mínimo hallado podemos pensarlo como el punto medio del último intervalo considerado.

Para terminar con el método del gradiente, resta analizar la última variante elegida: búsqueda lineal inexacta. Hay más de un método conocido para esta opción, en nuestro caso vamos a elegir el criterio de Armijo. Una de las diferencias más esenciales que presenta este método es que ya no vamos a utilizarlo explícitamente para minimizar φ, si no que diseñamos un algoritmo para, directamente, hallar un mínimo de f.

La idea general de Armijo es, a partir de valores fijos $\theta, \alpha \in (0,1)$, y de un punto inicial x_k, elegir una dirección d_k que cumpla:

$$\nabla f(x_k) d_k \leq -\theta \|\nabla f(x_k)\| \|d_k\| \tag{1}$$

Hecho esto, fijar un valor λ y reducirlo lo necesario hasta que $f(x_k + \lambda d) \leq f(x_k) + \lambda \alpha \nabla f(x_k) d_k^t$. Finalmente, considerar x_{k+1} como $x_k + \lambda d_k$ para volver a empezar.

En nuestro caso, tomamos $\lambda = 1$ y, en cada paso, lo reducimos multiplicandolo por 0,5. Además, elegimos d_k como $-\theta \nabla f(x_k)$, que cumple la condición correspondiente. En efecto:

$$\nabla f(x_k)d_k \leq -\theta\|\nabla f(x_k)\|\|d_k\|$$
$$\iff \nabla f(x_k).(-\theta)\nabla f(x_k) \leq -\theta\|\nabla f(x_k)\|\| - \theta\nabla f(x_k)\|$$
$$\iff -\theta\|\nabla f(x_k)\|^2 \leq -\theta\theta\|\nabla f(x_k)\|^2$$
$$\iff -\theta \leq -\theta^2$$
$$\iff \theta \geq \theta^2$$

Y esta última desigualdad es cierta dado que $\theta \in (0, 1)$.

1.2. Implementación

Estas cuatro maneras de implementar el método del gradiente quedan agrupadas bajo una única función, en el archivo "a1.m". Además de dos parámetros obvios (f, x_0), ésta recibe tres más: "búsquedaLineal", "opciones" y "gradiente".

El primero de éstos recibe valores entre 1 y 4, correspondiendo cada uno a la variante elegida para implementar el método:

1. Exacta de paso óptimo
2. Exacta de paso limitado
3. De orden 0
4. Inexacta: Armijo.

El vector opciones cuenta con cinco elementos: [MaxNumIter, tolGrad, gradHess, α, θ]. MaxNumIter espera que se indique la cantidad de iteraciones que se deben hacer. En tolGrad se debe indicar un valor suficientemente pequeño para el cual la norma del gradiente nos permita terminar de iterar. El tercer parámetro es binario y debe valer 0 en caso de que no se ingrese el gradiente y haya que calcularlo numéricamente, y 1 en caso contrario. α y θ serán los valores elegidos en caso de que la variante de búsqueda lineal que se quiera utilizar sea inexacta (Armijo).

Por último, el parámetro "gradiente" espera recibir la función que representa al gradiente de f en caso de que se haya escogido darlo como una entrada. Si no (es decir, si *gradHess* es 0), el programa hace caso omiso a este parámetro y calcula el gradiente numéricamente.

1.3. Método del gradiente con paso limitado

Nos detendremos ahora en el segundo caso de los mencionados en la sección anterior, el de búsqueda lineal exacta con paso limitado. Para comprender mejor su funcionamiento, dado $\delta > 0$, consideremos la siguiente función punto a conjunto:

$$\mathbf{S}^\delta(x, d) = \{y : y = x + \alpha d, 0 \leq \alpha \leq \delta; f(y) = \min_{0\leq\beta\leq\delta} f(x + \beta d)\}$$

Dada una dirección d y un punto x, $\mathbf{S}^\delta(x, d)$ mira a todos los puntos y tales que $y \in B[x, \delta]$ en la dirección d. Entre estos elementos, devuelve aquellos que alcancen el valor más pequeño de f.

Veamos que si f es continua, entonces $\mathbf{S}^\delta(x, d)$ es cerrada en (x, d).

Tomemos $(x_k, d_k) \to (x, d), (y_k)_k \subseteq \mathbb{R}^n$ tal que $y_k \in \mathbf{S}^\delta(x_k, d_k)$, y además $y_k \to y$. El objetivo es ver que $y \in \mathbf{S}^\delta(x, d)$. Como $y_k \in \mathbf{S}^\delta(x_k, d_k)$, tenemos que $y_k = x_k + \alpha_k d_k$, $0 \leq \alpha_k \leq \delta$ y $f(y_k) = \min_{0\leq\beta\leq\delta} f(x_k + \beta d_k)$.

Analicemos ahora el límite de α_k. Si asumimos que $d \neq 0$, entonces podemos afirmar que, a partir de un momento, $d_k \neq 0$. Luego:

$$\|y_k - x_k\| = \|\alpha_k d_k\| \Rightarrow \alpha_k = \frac{\|y_k - x_k\|}{\|d_k\|}$$

Así, $\alpha_k \to \frac{\|y-x\|}{\|d|} = \tilde{\alpha}$. Como teníamos que $0 \leq \alpha_k \leq \delta$, necesariamente $0 \leq \alpha \leq \delta$.

Además, $y_k \to y, d_k \to d, x_k \to x$, por lo que $y = x + \tilde{\alpha}d$. Sólo resta ver que $f(y) = \min_{0\leq\beta\leq\delta} f(x + \beta\delta)$. Pero sabemos que esto vale para y_k, es decir $f(y_k) \leq f(x_k + \beta d_k)\ \forall\ 0 \leq \beta \leq \delta$. Finalmente, como f es contínua, es inmediato que $f(y) \leq f(x + \beta d)\ \forall\ 0 \leq \beta \leq \delta$, que es lo que queríamos ver.

Para terminar, sólo falta analizar el caso $d = 0$. Sea entonces $(x_k, d_k) \to (x, 0), (y_k)_k \subseteq \mathbb{R}^n / y_k \to y, y_k \in \mathbf{S}^\delta(x_k, d_k)$. Queremos ver que $y \in \mathbf{S}^\delta(x, 0)$. Analicemos el conjunto $\mathbf{S}^\delta(x, 0)$.

$$\mathbf{S}^\delta(x,0) = \{y : y = x + \alpha 0, 0 \leq \alpha \leq \delta; f(y) = \min_{0 \leq \beta \leq \delta} f(x + \beta 0)\} = \{y : y = x; f(y) = f(x)\} = \{x\}$$

Luego, necesitamos ver que $y = x$. Como $y_k \in \mathbf{S}^\delta(x_k, d_k), y_k = x_k + \alpha_k d_k$, con $0 \leq \alpha_k \leq \delta$. Luego, tomando límite (recordemos que α_k está acotada), tenemos que $y = x + 0 = x$.

1.4. Experimentación y Resultados

1.4.1. Tiempos

En primer lugar, nos interesa saber si hay gran diferencia entre las variantes propuestas en cuanto a tiempos de corrida. Por otro lado, creemos que proveer explícitamente la función gradiente no solo aporta a la precisión sino también a los tiempos. Para medir esto, tomamos una función de ejemplo

$$f(x, y) = (x - y)^4 + 2x^2 + y^2 + -x + 2y$$

Y escribimos su gradiente de manera explícita. Corrimos entonces ambas variantes, con cada uno de los métodos de búsqueda lineal. Ahora, como los tiempos eran muy pequeños, decidimos correr 200 iteraciones de cada uno para amortizar posibles ruidos.

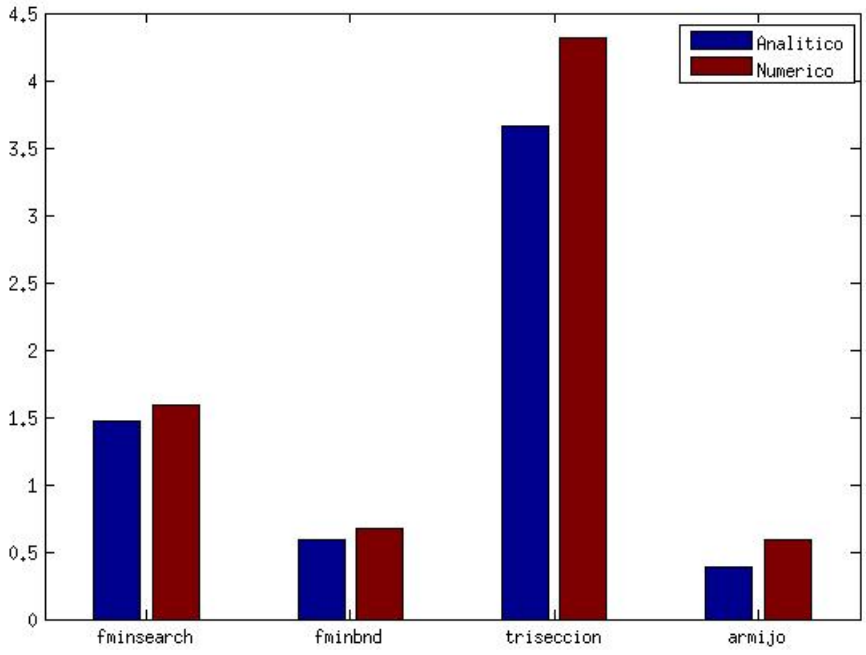

Figura 1: Tiempos del método del Gradiente, para la misma función, 200 iteraciones.
En azul, con la función gradiente provista. En marrón, calculándola numéricamente.

Vemos que hay una diferencia consistente entre usar el gradiente o calcularlo, siendo este último más costoso. Por otro lado, al comparar los métodos de búsqueda lineal, trisección resulta claramente el más lento, estando fminbnd y Armijo por debajo en tiempos. El más rápido de todos terminó siendo Armijo, incluso casi tres veces más rápido que el fminsearch de Matlab, lo cual nos sorprendió bastante.

1.4.2. Precisión

Además, nos interesa saber cuánto se aproxima cada uno de ellos al mínimo "real". Para ello, calculamos un mínimo local cerca de $x_0 = (1, 1)$ (o mejor dicho, le pedimos a Matlab que nos dé su mejor aproximación del mismo) con el comando fminsearch, y luego tomamos la norma de la diferencia. Para cada método, estos son los resultados:

Método	Norma del error
fminsearch	0.0001
fminbnd	0.0001
Trisección	0.0021
Armijo	0.0002

Vemos que Trisección no solamente resulta más lento, sino también más impreciso. Por otro lado, Armijo resulta más rápido pero no tan preciso como las búsquedas lineales de Matlab.

1.4.3. Otras funciones

Todos los tests anteriores se realizaron usando la misma función. ¿Qué pasará si usamos una distinta? Para responder esa pregunta, tomamos varias funciones conocidas y realizamos el mismo análisis sobre ellas. La única diferencia será que ya no sabemos la función gradiente de ellas, por lo que solo queda la opción de calcularlos numéricamente.

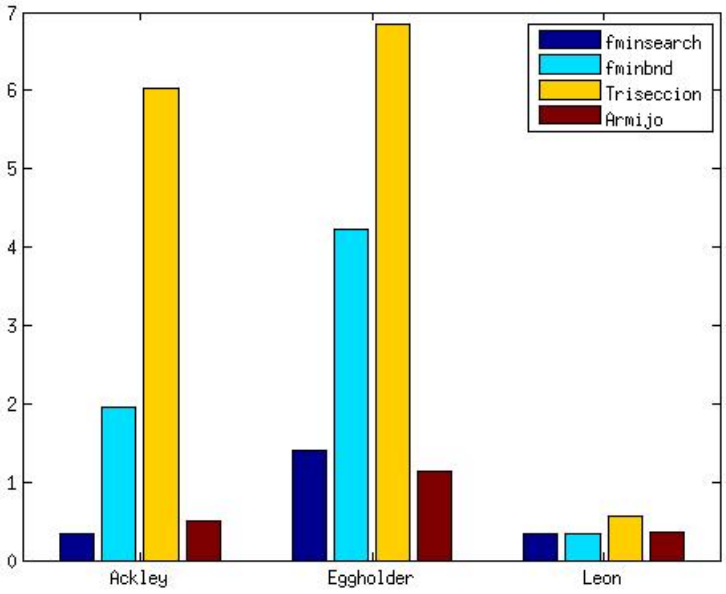

Figura 2: Tiempos del método del Gradiente, para tres funciones de ejemplo, con las cuatro variantes. Para las dos primeras funciones, 100 iteraciones, y para la última 1000.

Acá vemos que en dos de las tres funciones, Armijo funcionó más lenta que fminsearch (lo que no sucedía con nuestra función). Queda pendiente ver qué pasa en otros casos.

2. Ejercicio 2: Método del Gradiente Conjugado

2.1. Desarrollo

El método del Gradiente Conjugado será implementado para funciones cuadráticas y a la vez generalizado para no cuadráticas.

Para el primer caso, $f(x) = \frac{1}{2}x^tQx - b^tx$, con Q el hessiano de f (que es constante) y b el gradiente en $(0, 0)$. La idea general es, partiendo de un x_0 inicial, tomar $d_0 = -\nabla f(x_0)$, que justamente, por la naturaleza de la función, es $b - Qx_0$. Luego, se construye $(x_k)_{k=0,\dots,n}$ recursivamente:

- $x_{k+1} = x_k + \alpha_k d_k$ con

$$\alpha_k = \frac{-g_k^t d_k}{d_k^t Q d_k}$$

donde, nuevamente por ser una función cuadrática, $g_k = Qx_k - b$

- $d_{k+1} = -g_{k+1} + \beta_k d_k$ con

$$\beta_k = \frac{g_{k+1}^t Q d_k}{d_k^t Q d_k}$$

Vimos en la teórica que el método converge en a lo sumo n pasos, pues en cada uno minimizamos sobre un subespacio de una dimensión más grande que el anterior.

Ahora bien, cuando la función no es cuadrática, hay que modificar un poco el algoritmo. Los gradientes g los vamos a tener que calcular numéricamente, y donde usamos Q tendremos que usar el Hessiano de f en x_k.

2.2. Experimentación

2.2.1. Analítico vs Numérico

Mencionamos que tenemos dos opciones para obtener el Gradiente y el Hessiano de la función: recibirlos como parámetros o calcularlos numéricamente. Podemos comparar, entonces, la performance de ambos en cuanto a tiempos. Como terminan muy rápidamente, de nuevo tomamos 200 iteraciones de cada método. Además, como sólo estamos obteniendo dos números, es difícil compararlos, por lo que realizamos el mismo experimento 25 veces y graficamos los resultados.

La función que usamos de ejemplo es

$$f(x, y) = x^2 + 3y^2 + 2xy + x + 3y$$

Por lo que si llamamos

$$Q = \begin{bmatrix} 2 & 2 \\ 2 & 6 \end{bmatrix}; \quad b^t = (1, 3)$$

Tenemos que

$$f(x, y) = \frac{1}{2}[x, y] Q \begin{bmatrix} x \\ y \end{bmatrix} - b^t \begin{bmatrix} x \\ y \end{bmatrix}$$

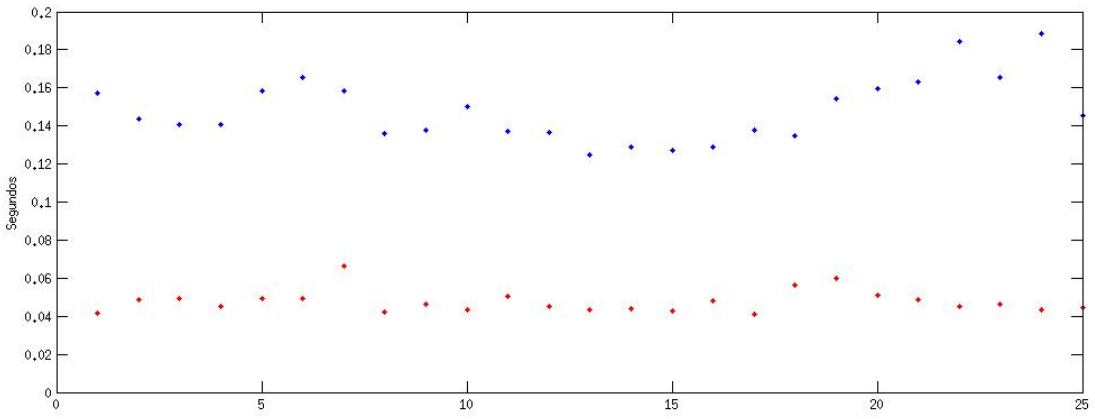

Figura 3: Tiempos del método del Gradiente Conjugado. Cada punto azul representa 200 iteraciones del método calculando numéricamente el gradiente y el hessiano. Cada punto rojo, recibiendo ambos como parámetros.

Se ve claramente que proporcionar gradiente y Hessiano mejora bastante la performance del método.

3. Ejercicio 3

3.1. Desarrollo

El ejercicio 3 proponía buscar un método o variante distinta a las propuestas anteriormente. Decidimos utilizar el del gradiente, pero utilizando el método de la secante para encontrar el paso óptimo en la dirección d_k.

El método de la secante sirve para encontrar ceros de funciones. Ahora bien, en el minimizador buscado de φ vale que su derivada es 0, por lo que podemos aplicar el método a $\psi = \varphi'$.

Dados puntos iniciales x^0 y x^1, se calcula la recta que une $\psi(x^0)$ con $\psi(x^1)$. Luego, se toma al siguiente elemento como la intersección entre dicha recta y el eje x. El algoritmo se repite, siempre tomando los dos últimos puntos considerados, y bajo ciertas condiciones se converge a un mínimo local. Es importante notar que el éxito del algoritmo depende fuertemente de la elección de un buen punto inicial.

Como dato adicional, nosotros buscamos un mínimo en el intervalo $(0, +\infty)$, por lo que decidimos truncar los x_k negativos a cero. Decidimos, además, parar si los dos últimos puntos se encuentran a una distancia muy pequeña.

Por las características de la función, esperaremos que el punto obtenido sea efectivamente un mínimo de φ.

3.2. Tiempos

Como Secante es un método de búsqueda lineal, vamos a compararlo contra los propuestos en el ejercicio 1. Nuevamente proponemos variantes Analítica y Numérica, y tanto la f como su gradiente son los mismos. Veamos qué sucede.

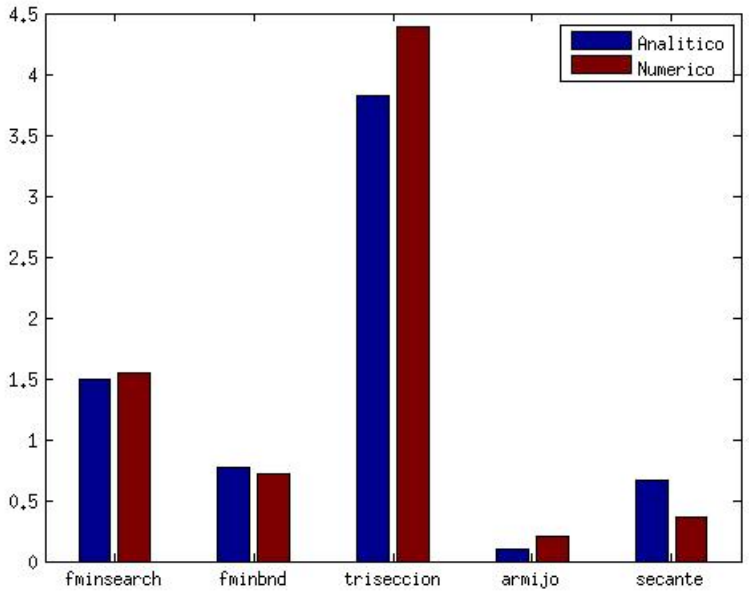

Figura 4: Tiempos del método del Gradiente, para la misma función, 200 iteraciones.
En azul, con la función gradiente provista. En marrón, calculándola numéricamente.

Se puede observar que, si bien no performa como Armijo, sí tiene un buen rendimiento en comparación con las rutinas de Matlab.

Es importante notar, además, que Secante es muy sensible al dato inicial. Solo funciona bajo ciertas condiciones de la f, y en un entorno de un mínimo local, si estas condiciones no se cumplen el método diverge. Al comprobar que efectivamente llega a un mínimo, nos restringimos a funciones conocidas, y con el x_0 lo suficientemente cercano al minimizador local buscado.

4. Anexo: Implementación numérica del gradiente y hessiano

Como ya mencionamos anteriormente, en todos los algoritmos implementados existe la opción de ingresar el gradiente y/o hessiano que corresponde, o simplemente no darlos como parámetro y dejar que el cálculo se efectúe numéricamente. Para poder hacer esto posible, construimos dos funciones auxiliares "gradNum.m" y "hessNum.m"

4.1. Gradiente

Recordemos que, dada una función $f : \mathbb{R}^n \to \mathbb{R}$, sus derivadas parciales vienen dadas por:

$$\frac{df}{dx_i}(x_1, \dots x_n) = \lim_{h \to 0} \frac{f(x_1, ..., x_i + h, ..., x_n) - f(x_1, ..., x_i, ..., x_n)}{h}$$

Luego, para implementar un algoritmo que calcule el gradiente de f en un punto x, simplemente consideramos un valor de h suficientemente chico (en nuestro caso, elegimos $h = 0{,}00001$), y realizamos la cuenta mencionada para cada variable.

4.2. Hessiano

Para el caso del Hessiano, la forma que elegimos de calcularlo es similar a la del gradiente, pero lleva un poquito más de trabajo.

Supongamos que queremos calcular $\frac{d^2 f}{dx_i dx_j}$; podemos pensar que nuestro objetivo es hallar $\frac{d}{dx_j}(\frac{df}{dx_i})$. Luego, como en la sección anterior ya vimos una forma de aproximar numéricamente $\frac{df}{dx_i}$, podríamos copiarnos de esta idea para también aproximar $\frac{d}{dx_j}(\frac{df}{dx_i})$. Es decir, tomando un h pequeño:

$$\frac{d}{dx_j}(\frac{df}{dx_i}) \approx \frac{d}{dx_j}(\frac{f(x_1, ..., x_i + h, ..., x_n) - f(x_1, ..., x_i, ..., x_n)}{h}) \approx$$

$$\frac{1}{h}[\frac{d}{dx_j}(f(x_1, ..., x_i + h, ..., x_n)) - \frac{d}{dx_j}(f(x_1, ..., x_i, ..., x_n))] \approx$$

$$\frac{1}{h}[\frac{(f(x_1, ..., x_i + h, ..., x_j + h, ..., x_n) - f(x_1, ..., x_i + h, ..., x_n)}{h} - \frac{f(x_1, ..., x_j + h, ..., x_n) - f(x_1, ..., x_j, ..., x_n)}{h}] \approx$$

$$\frac{f(x_1, ..., x_i + h, ..., x_j + h, ..., x_n) + f(x_1, ..., x_i, ..., x_j, ..., x_n) - f(x_1, ..., x_i + h, ..., x_n) - f(x_1, ..., x_j + h, ..., x_n)}{h^2}$$

Finalmente, como la matriz Hessiana en cada lugar (i, j) lo que tiene es $\frac{d^2 f}{dx_j dx_i}$, sólo hizo falta saber ubicar estas aproximaciones en el lugar correspondiente.